Tutorial 4: TCP Traffic Analysis

Objective: Go through P2

P2 Goal

You are required to write a python program to analyze the TCP protocol behavior

- Understand the details of state management in Transmission Control Protocol (TCP)

Requirements

- You will be given a sample TCP trace file (sample-capture-file.cap)
  - The trace file records a single server accessing different websites
  - TA may use a different trace file to test your code
- Your program needs to parse and process the trace file, and track TCP state information
  - A TCP connection is identified by a 4-tuple (IP source address, source port, IP destination address, destination port)
  - Packets can flow in both directions on a connection
  - Packets from different connections can be arbitrarily interleaved with each other in time

Summary for Each Connection

- The state of the connection
  - For example, S0F0 (no SYN and no FIN), S1F1 (1 SYN and 1 FIN), and S3F1 (3 SYN and 1 FIN), etc.
  - For consistence, we count a SYN+ACK segment as a SYN message
- The starting time, ending time, and duration of each complete connection
- The number of packets sent in each direction on each complete connection, as well as the total packets
- the number of data bytes sent in each direction on each complete connection, as well as the total bytes.

Summary Statistical Results for All Connections

- The number of reset TCP connections observed in the trace
- The number of TCP connections that were still open when the trace capture ended
- The number of complete TCP connections observed in the trace
- Regarding the complete TCP connections you observed:
  - The minimum, mean, and maximum time durations of the complete TCP connections
  - the minimum, mean, and maximum RTT (Round Trip Time) values of the complete TCP connections
  - the minimum, mean, and maximum receive window sizes (both sides) of the complete TCP connections.

Input Format

- Input format
  - $ python3 yourcode.py tracefile.cap

Output Format

A) Total number of connections:

B) Connections' details:

Connection 1:
Source Address:
Destination address:
Source Port:
Destination Port:
Status:
(Only if the connection is complete provide the following information)
Start time:
End Time:
Duration:
Number of packets sent from Source to Destination:
Number of packets sent from Destination to Source:
Total number of packets:
Number of data bytes sent from Source to Destination:
Number of data bytes sent from Destination to Source:
Total number of data bytes:
END

…

Connection N:

……

C) General
Total number of complete TCP connections:
Number of reset TCP connections:
Number of TCP connections that were still open when the trace capture ended:

D) Complete TCP connections:

Minimum time duration:
Mean time duration:
Maximum time duration:
Minimum RTT value:
Mean RTT value:
Maximum RTT value:
Minimum number of packets including both send/received:
Mean number of packets including both send/received:
Maximum number of packets including both send/received:
Minimum receive window size including both send/received:
Mean receive window size including both send/received:

Maximum receive window size including both send/received:

Deliverables and Marking Scheme

- Zip your assignments (code) as one tar file using %tar -czvf on linux.csc.uvic.ca.

Components	Weight
Total number of connections	25
Connections’ details	30
General Statistics	20
Complete TCP connections:	20
Readme.txt, code style	5
Total Weight	100

Plagiarism

This assignment is to be done individually.

You are encouraged to discuss the design of your solution with your classmates, but each person must implement their own assignment.

Notes

- Your code will be tested on the server linux.csc.uvic.ca
- Use python3 packages supported by linux.csc.uvic.ca
- Packages that can automatically extract each packet are not allowed to be used, e.g.,

  scapy, which contains methods such as RawPcapReader

  python-libpcap

  pyshark

  pycapfile

  pypcap

  …

Hints – The Structure of Cap Files

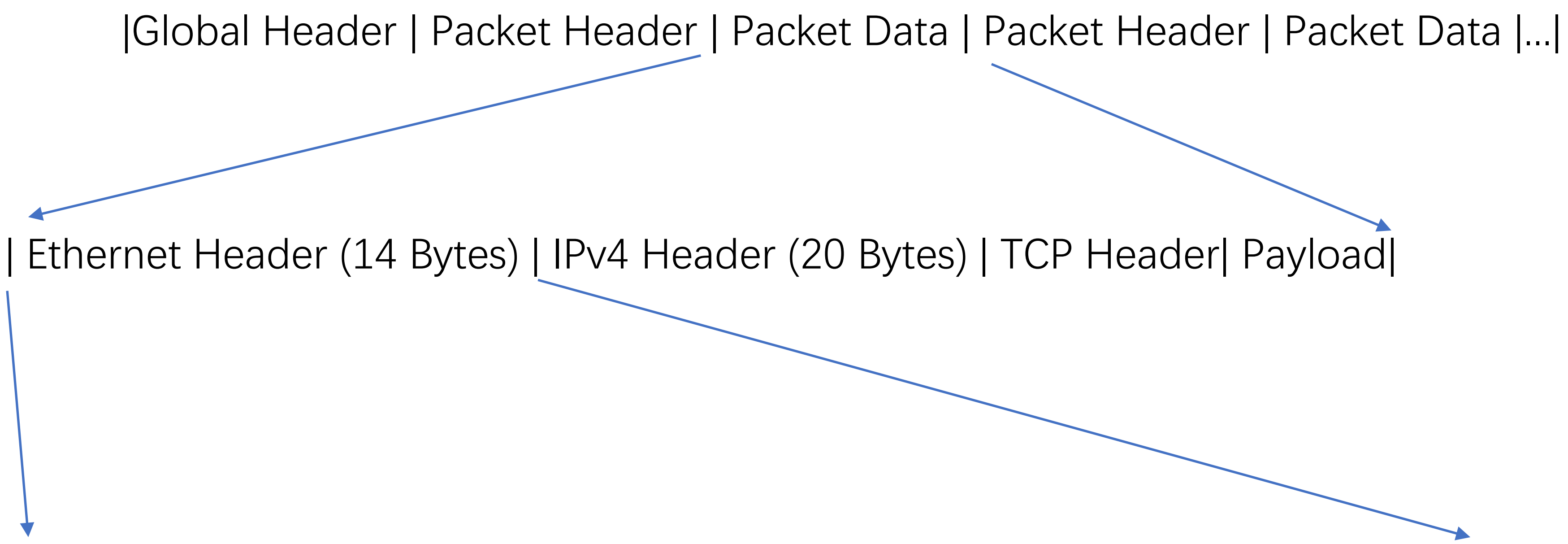
https://gist.github.com/unitycoder/a82365a93c9992f7f9631741fe007e9d

|Global Header | Packet Header | Packet Data | Packet Header | Packet Data |...|

bytes	type	Name	Description
4	uint32	magic_number	'A1B2C3D4' means the endianness is correct
2	uint16	version_major	major number of the file format
2	uint16	version_minor	minor number of the file format
4	int32	thiszone	correction time in seconds from UTC to local time (0)
4	uint32	sigfigs	accuracy of time stamps in the capture (0)
4	uint32	snaplen	max length of captured packed (65535)
4	uint32	network	type of data link (1 = ethernet)

|Global Header | Packet Header | Packet Data | Packet Header | Packet Data |...|

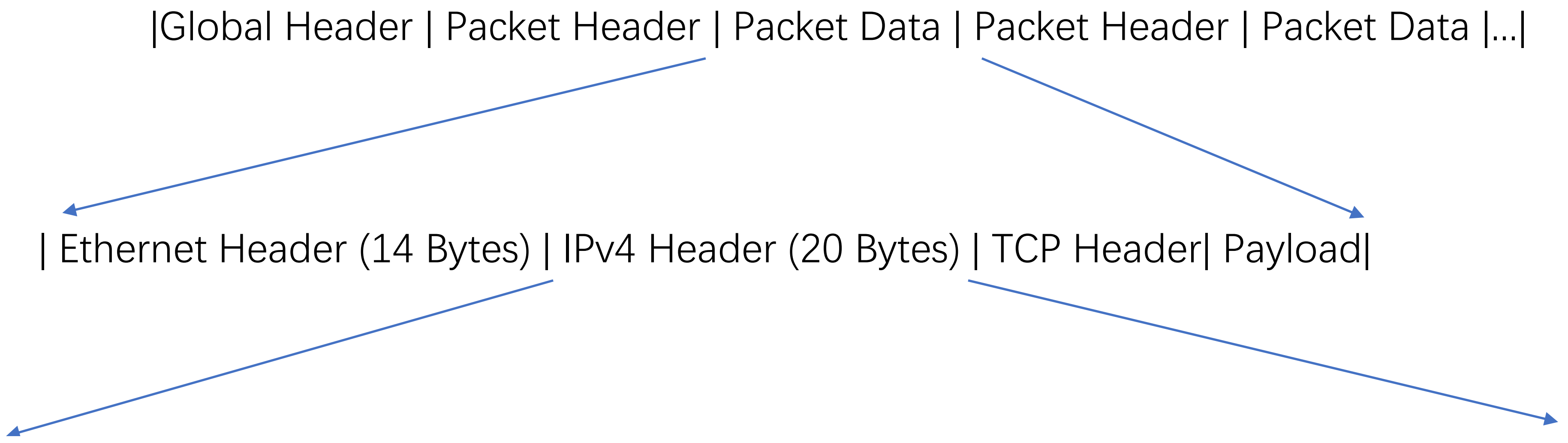
bytes	type	Name	Description
4	uint32	ts_sec	timestamp seconds (number of seconds since the start of 1970)
4	uint32	ts_usec	timestamp microseconds
4	uint32	incl_len	contains the size of the saved packet data in our file in bytes (following the header)
4	uint32	orig_len	actual length of packet



|Global Header | Packet Header | Packet Data | Packet Header | Packet Data |…|

| Ethernet Header (14 Bytes) | IPv4 Header (20 Bytes) | TCP Header| Payload|

bytes	**Name**
6	Destination MAC address
6	Source MAC address
2	Ethernet Type



|Global Header | Packet Header | Packet Data | Packet Header | Packet Data |…|

| Ethernet Header (14 Bytes) | IPv4 Header (20 Bytes) | TCP Header| Payload|

4	IP Version Number (4)
4	IHL
8	Type of Service
16	Total Length
16	Identification
4	Flags
12	Fragment Offset
8	Time to Live
8	Protocol
16	Header Checksum
32	Source Address
32	Destination Address

How to Extract Structures?

- Use package **struct**
- Interpret bytes as packed binary data
- Example

```
>>> from struct import *
>>> pack('hhl', 1, 2, 3)
b'\x00\x01\x00\x02\x00\x00\x00\x03'
>>> unpack('hhl', b'\x00\x01\x00\x02\x00\x00\x00\x03')
(1, 2, 3)
>>> calcsize('hhl')
8
```

Format	C Type	Python type	Standard size
h	short	integer	2
L	unsigned long	integer	4

Examples of IP header

- The code for basic structures (packet, IP and TCP) will be posted in brightspace later.

```
class IP_Header:
    src_ip = None #<type 'str'>
    dst_ip = None #<type 'str'>
    ip_header_len = None #<type 'int'>
    total_len = None    #<type 'int'>

    def __init__(self):
        self.src_ip = None
        self.dst_ip = None
        self.ip_header_len = 0
        self.total_len = 0
```

```python
def ip_set(self,src_ip,dst_ip):
    self.src_ip = src_ip
    self.dst_ip = dst_ip

def header_len_set(self,length):
    self.ip_header_len = length

def total_len_set(self, length):
    self.total_len = length

def get_IP(self,buffer1,buffer2):
    src_addr = struct.unpack('BBBB',buffer1)
    dst_addr = struct.unpack('BBBB',buffer2)
    s_ip = str(src_addr[0])+'.'+str(src_addr[1])+'.'+str(src_addr[2])+'.'+str(src_addr[3])
    d_ip = str(dst_addr[0])+'.'+str(dst_addr[1])+'.'+str(dst_addr[2])+'.'+str(dst_addr[3])
    self.ip_set(s_ip, d_ip)
```